



Comal og alle de andre.

Head gør man, når de eksisterende programmeringssprog ikke er gode nok til formålet?

Dette problem var lektor Børge Christensen og Benedicte Løfstedt stillet overfor, da de skulle undervise i programmering på Tønder Amtsgymnasium.

Der findes mange programmeringssprog af varige mellem. Nogle er decideret egnet til undervisning, f.eks. BASIC, LOGO og PASCAL, - andre er bedre egnet til beregninger, f.eks. FORTRAN og COBOL, mens LISP og PROLOG er bedst egnet til f.eks. forskning i kunstig intelligens (Artificial Intelligence).

Der skulle jo så være nemt at finde et brugbart programmeringssprog med alle de valgmuligheder. Der er jo flere hundrede forskellige computersprog. Nej, herre herre, nemt er det bestemt ikke! Faktisk er det ganske forvirrende, og det gør ikke sagen bedre, at hver eneste programmer har sine bestemte idier om, hvad det er bedst. Bare sproget BASIC findes i flere tusinde dialekter, hvor med deres specialiteter, der også gør det umuligt at indtaste et BASIC program fra en anden computer uden at ændre kendskab til computeren.

Tiden anvender også på flere osv., og de fleste programmeringssprog har skellinger de på bagen. BASIC er over 10 år gammel - og det kan end IEDB allerede bemærke var det ikke plads til «pære» programmering - i løbet af IEDB maskinen var simpelthen for dyrt. Hvis det var muligt, gentogte man en programtæmp til hushænde. Det indiker bl.a. at programmer var sammensat af en masse GOTO's. I dag er logikretorik blevet så billige, at man har skullet af fiktive. Dette afvikles kan bruges fornuftigt. Men er det ikke mere nødvendigt at klare programmerne sammen med GOTO statements. Med mere lagerplads kan den såkaldte spaghetti-programmering undgås. Programmerne kan struktureres til program, og dermed samle de dele af programmeret, der hører sammen til en helhed - også selv om der kunne spares 20 linier ved at bruge en enkelt GOTO. Det gør programmerne nemme at overvåge, rette og vedligeholde.

Det bedste af to verdener

Men hvorfor er BASIC så ikke godt nok? BASIC er et blødt og let tilgængeligt sprog som mange lærer, men det har en fundamental alene: det er påklæbet struktureret. Et sprog som Pascal dermed er (hvis kan være) yderst struktureret, men det er ikke nemt at gå til, specielt ikke for en nybegynder. Men hvad med at blande de bedste sider af disse to meget populære sprog, så det er både let at gå til og samtidig understøtte struktureret programmering? Det var præcis hvad Benedicte Løfstedt og Børge Christensen gjorde, og resultatet blev: COMAL. Også COMAL benytter

COMBosn Algorithmic Language. COMAL ligner Pascal og BASIC på mange punkter, men det er vist forskellige.

Prøv angiv at kigge på disse 3 programmer. De udfører nøjagtig det samme, men de er skrevet i hhv. BASIC, Pascal og COMAL:

```
10 REM Dette BASIC program udfører 9-tabellen
20 DIM I(10)
30 I = 0
40 PRINT " "
50 I = 1
60 PRINT "1 gange 9 er " I
70 I = 2
80 PRINT "2 gange 9 er " I
90 I = 3
100 PRINT "3 gange 9 er " I
110 I = 4
120 PRINT "4 gange 9 er " I
130 I = 5
140 PRINT "5 gange 9 er " I
150 I = 6
160 PRINT "6 gange 9 er " I
170 I = 7
180 PRINT "7 gange 9 er " I
190 I = 8
200 PRINT "8 gange 9 er " I
210 I = 9
220 PRINT "9 gange 9 er " I
230 END
```

```
program Pascal
10 REM Dette Pascal program udfører 9-tabellen
20 DIM I(10)
```

```
var
  I, n: integer;
```

```
label
  n1;
```

```
begin in n1, label n1
```

```
  I := 0;
```

```
  n1;
```

```
  writeln(' 9 gange 9 er ' & I);
```

```
  repeat n1
```

```
  if I < 9 then I := I + 1; then goto n1
```

```
  writeln('End of Pascal 9-table');
```

```
end. in n1, label n1
```

```
10 REM Dette COMAL-program udfører 9-tabellen
20 IF I(10) THEN GOTO "9"
30 I = 0
40 PRINT " "
50 I = 1
60 PRINT "1 gange 9 er " I
70 I = 2
80 PRINT "2 gange 9 er " I
90 I = 3
100 PRINT "3 gange 9 er " I
110 I = 4
120 PRINT "4 gange 9 er " I
130 I = 5
140 PRINT "5 gange 9 er " I
150 I = 6
160 PRINT "6 gange 9 er " I
170 I = 7
180 PRINT "7 gange 9 er " I
190 I = 8
200 PRINT "8 gange 9 er " I
210 I = 9
220 PRINT "9 gange 9 er " I
230 END
```

Programmerne udfører 9-tabellen, en linie ad gangen. Først 9 gange 9, derefter 1 gange 9, 2 gange 9 osv. osv., indtil man når 9 for 9.

BASIC programmet indledes med en 2-liniers kommentar. Derefter initialiseres variablen «I». I BASIC er det ikke nødvendigt at erstatte sine variable, så variablen «I» er automatisk nul. Derefter går der dette ikke i alle BASIC udgaver, så det er bedst at gøre det til en regel altid at erstatte sine variable.

PRINT sætningen udfører en linie af gangstabellen. Variablen «I» og «I» er tilfældige nye værdier. Spørgsmålet udføres på skærmen og svaret hentes ind i computeren til bearbejdnings. Hvis brugeren indtaster andet end «I» eller «N» hop-

per programmet tilbage til linie 40. Hvis betingelsen er sand (der er trykket på N) udføres en afsluttende bemærkning, og programmet stopper.

PASCAL programmet er noget mere omfattende. Programmet skal først læses af masken; derefter følger de 2 linier med kommentarer. Alle variable skal defineres i Pascal, så et *n* defineres som heltal (integer) og *v*-variablen defineres som et bogstav (char).

Liniesamner har ingen betydning i Pascal. Dette indebærer at man, for at læsne komens ordning i programmet, må hoppe til en skudslet linie! Den skal også defineres. Endnu Pascal-programmet der var en ledetdefinition, men i dette lille program, vil gøre højtlyt og fortælle mig, hvilken dumhed jeg har gjort. Men herom senere!

Hovedprogrammet starter med «BEGIN» og afsluttes med «END» - eller ligner dette i øvrigt BASIC-programmet. PRINT hedder WRITE, INPUT hedder READLN og tildelinger (*n* = 0; *n* = 0) bruges : = i stedet for en lighedstegnelse. Lighedstegnet alone bruges kun ved sammenligninger i Pascal.

Til side COMAL programmet. Kommentarer i COMAL indskrives af // - det er nemmere at se end BASIC's Rem og * (opstart).

Det er ikke nødvendigt at angive en ledetvariabel størrelse, men i COMAL kan den få pascals det antal index og den længde der skal have, på følgende måde:

```
DIM a$ OF 1,64(1) OF 3,4(5,2) OF 200,d$ (2000:1002) OF 2000
```

Variablen a\$ er på 1 tegn.

Variablen d\$ er et array med 2 elementer: d\$(0) og d\$(1) på 3 tegn.

Variablen c\$ er et to-dimensionelt array på 200 tegn, og Variablen d\$ er et array med 3 elementer: d\$(000), d\$(001) og d\$(002), på hver 2000 tegn.

I programkødet er variablen c\$ kun på et enkelt tegn, dvs. at «n», «M» og «P» alle accepteres som værende lig med n, m, p, så c\$ kan indeholde det første tegn, der indtastes.

Alle variable skal tildelen en værdi inden de bruges i COMAL. Der anvendes de samme tildelingsregler som i Pascal, og ligesom Pascal kan COMAL heller ikke henvises til liniesamner - derfor må vi benytte en label der hedder «n» (Komma (,) og semikolon (;) har lidt forskellig betydning i BASIC og COMAL's PRINT sætninger. Det vil vi beskræftige os mere med i næste artikel.

COMAL's INPUT-sætninger kan udførelse ledetlister (f.eks. «MERE H/VN»), som logisk hører sammen med INPUT-sætninger. Det samme gælder for «END», der kan udføres en afsluttende bemærkning.

Det er ikke muligt at lave indrykninger i et COMAL program. Det skyldes, at COMAL-fortolken fjerner alle unødvendige mellemrum!

Da har måske studiet over linie 6070, hvor der findes en GOTO rø. Det er også en provokation for min side. COMAL ruder over en forfærdelig løbestrutur, så den valgte løsning er faktisk den stærkeste løsning! Nu skal du se, hvor let løsningsprogrammet bliver, når vi bruger en løkke i stedet.

REPEAT/UNTIL løkken gentager programkødet, der ligger inden løkken indtil betingelsen efter UNTIL-sætningen er sand. Et eksempel:

```
a := a + 1
UNTIL a > 9
```

I eksemplet har vi en tæller *a*, som tælles op i REPEAT/UNTIL løkken indtil den er større end 9. Løkken gentages altså 10 gange. En REPEAT/UNTIL løkke gennemløbes altid mindst en gang, da betingelsen ligger i slutningen af løkken.

Det modsatte gælder for WHILE/ENDWHILE. En WHILE/ENDWHILE løkke gentages så længe betingelsen efter WHILE er sand:

```
a := 0
WHILE a < 9
a := a + 1
ENDWHILE
```

Så længe *a* er mindre end 10, tælles *a* op med en - det sker i alt 10 gange i dette eksempel. Hvis WHILE-betingelsen er falsk (= forsket) fra starten, bliver løkken slet ikke udført, da betingelsen bliver testet allerede inden løkken starter.

Endelig har vi den almindelige FOR/ENDFOR løkke. Det er den, der hedder FOR/NEXT i BASIC.

```
FOR a := 1 TO 10 DO
ENDFOR a
```

Denne løkke udføres også 10 gange, med *a* som tæller-variabel.

While og REPEAT bruges, når det endelige antal iteration (dvs. antal gange løkken skal gennemløbes) er kendt. I stedet har man ofte en betingelse, der fortæller, om løkken er færdig. FOR/ENDFOR bruges da stedet, hvor antallet af gennemløb er kendt inden løkken endes.

COMAL har også «endelige løkker». LOOP/ENDLOOP har hverken test eller tællervariabel, men man kan komme ud af løkken på forskellig måde alligevel. Enten ved hjælp af EXIT eller TIMES. Med EXIT kan man lave en betingelse midt i løkken der, hvis den bliver sand, får programmet til at hoppe ud af løkken. Med TIMES kan man, uden betingelser og tællervariabel, få løkken udført et forud bestemt antal gange. Et eksempel på dette:

```
LOOP 10 TIMES
NULL
ENDLOOP
```

Linien NULL bliver udført 10 gange. NULL er et COMAL ord, der ikke udføres noget som helst.

Efter denne gennemgang af forskellige muligheder for løkker, kan COMAL programmet skrives i sin endelige form:

```
0010 // Helt lille program udfører 10-løkkeren
0020 // Indtast det første tal, der skal
0030 DIM a$ OF 1
0040 INPUT a$
0050 REPEAT
0060 PRINT a$, " gange "
0070 a := a + 1
0080 UNTIL a > 10
0090 END
```

Hæftet jo var meget pæntret ud.

I næste artikel gennemgås flere af COMAL's specialde kommandoer.

Fortsættes side 65

```
a := 0
REPEAT
```